

**METHOD AND APPARATUS FOR GENERIC SEARCH INTERFACE  
ACROSS DOCUMENT TYPES**

**Inventors: Muljadi Sulistio**

**Yang Wei**

**Kelly Lane Schwarzhoff**

**Yuan Din**

**Sun Lee**

**Andy Yang**

**CROSS REFERENCE TO RELATED APPLICATIONS**

**[0001]** The present application is related to commonly assigned, co-pending U.S. patent application No. 09/794,302, filed on February 27, 2001, entitled A METHOD AND APPARATUS FOR VIEWING ELECTRONIC COMMERCE-RELATED DOCUMENTS (CMRC 1002-1), by inventors Andrew Everett Davidson, Kelly Lane Schwarzhoff, Gunawan Herri, Changyi Zhu, Ari Krish, Muljadi Sulistio, and Sun Keun Lee, which is hereby incorporated by reference as if fully set forth.

**[0002]** This application is further related to the following commonly assigned, co-pending applications filed on December 18, 2001, the same day as the present application: U.S. patent application No. \_\_\_\_, xCBL MAILBOX METHODS AND DEVICES (CMRC 1010-1), by inventors Muljadi Sulistio, Yang Wei, Kelly Lane Schwarzhoff, Yuan Din, Sun Lee and Andy Yang; U.S. patent application No. \_\_\_\_, METHOD AND APPARATUS FOR DECLARATIVE UPDATING OF SELF-DESCRIBING, STRUCTURED DOCUMENTS (CMRC 1007-1), by inventors Muljadi Sulistio, Yang Wei, Kelly Lane Schwarzhoff and Yuan Din; and U.S. patent application No. \_\_\_\_, METHOD AND APPARATUS FOR DECLARATIVE ERROR HANDLING AND PRESENTATION (CMRC 1008-1), by inventors Muljadi Sulistio, Yang Wei, Kelly Lane Schwarzhoff and Yuan Din. These applications filed on the same day are hereby incorporated by reference as if fully set forth.

## COPYRIGHT NOTICE

[0003] A portion of the disclosure of this patent document contains material, which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure is it  
5 appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

## BACKGROUND OF THE INVENTION

[0004] Electronic commerce between businesses has gained substantial momentum. Electronic marketplaces and XML or similar documents have begun to  
10 replace traditional EDI formats for commerce-related documents. Still, many businesses, particularly small and medium-sized businesses, have not adopted automated EDI or XML document processing. Whatever their size, many businesses face the prospect of an expedited implementation of EDI or XML document  
15 processing. It remains easier for large trading partners to generate XML or similar documents than it is for small to medium-sized businesses to adopt the technology needed to process them. In addition, a full scale conversion to EDI or XML transaction processing may involve far more documents than a business can practically convert in a workable time frame or on a reasonable budget.

[0005] One problem with the implementation of EDI or XML transaction  
20 processing is the complexity and cost of procedural programming to process business documents. Procedural programming, otherwise known as hard coding, requires much effort to describe document transformations and manipulations in procedural terms, using programming languages such as Java and C++. This effort translates into time for implementation and cost of implementation.

[0006] In some domains or problem spaces, declarative programming has been  
25 introduced. It is generally hoped that so-called declarative programming can make program customization accessible even to non-programmers. At the same time, it has been recognized that declarative programming is best when applied to a limited domain. Accordingly, declarative approaches are narrow and tailored, not generally  
30 applied.

[0007] Therefore, in the domain of exchanging self-defining, structured documents, it is desirable to develop declarative methods and components for

simplifying the handling of documents. Declarative methods and components can improve interactions with users, particularly in the areas of producing documents, presenting error messages and searching for documents.

## SUMMARY OF THE INVENTION

- 5 [0008] The present invention includes a method and device for generic searching across document types. A further aspect of the present invention is indexing and searching XML documents using an HTML interface generated by a declarative transformation. Additional aspects of the present invention are described in the claims, specification and drawings.

## BRIEF DESCRIPTION OF FIGURES

10

[0009] Figure 1 is an overview of the user interface for an xCBL mailbox.

[0010] Figure 2 is a sample screen display of a folder list, as part of a user interface.

15

[0011] Figure 3 is a sample screen display for a list of documents within a folder, as part of a user interface.

[0012] Figure 4 is a sample screen display for indicating a reply document type.

[0013] Figure 5 is the sample screen display for searching on a selected document type, as part of a user interface.

20

[0014] Figure 6 is a block diagram of activities involving two trading partners communicating through a document exchange system.

[0015] Figure 7 is an application flow providing additional detail of activities involving two trading partners communicating through a document exchange system.

[0016] Figure 8 is a block diagram of a self-describing, structured document.

[0017] Figure 9 illustrates participants in a marketplace.

25

[0018] Figure 10 is a database schema for a data structure that may be used to practice aspects of the present invention.

[0019] Figure 11 is one architecture for software components that handle generic documents.

30

[0020] Figure 12 is a user interface and processing diagram, including validation of user created documents.

[0021] Figure 13 is a user interface for replying to a request for quote with a quote.

[0022] Figure 14 is an action sequence for replying to a document.

[0023] Figure 15 is a sequence diagram illustrating use of a servlet to control responses to a user action request.

[0024] Figure 16 is a sequence diagram illustrating interaction between a servlet, interpreter and data structure engine, in response to data from a user.

[0025] Figure 17 is a block diagram illustrating posting of data and some of the components involved in responding.

[0026] Figure 18 is a simplified flow chart for a method of updating a self-describing, structured document.

[0027] Figure 19 illustrates a protocol for updating a self-describing, structured document corresponding to one of a plurality of document schemas.

[0028] Figure 20 is a flowchart overview of validation.

[0029] Figure 21 is a flowchart overview of a method of searching a plurality of self-describing structure documents.

[0030] Figure 22 a flowchart overview of preparing documents for searching.

[0031] Figures 23-26 illustrate data tables used to prepare documents for searching.

### DETAILED DESCRIPTION

[0032] The following detailed description is made with reference to the figures.

Preferred and alternative embodiments are described to illustrate the present invention, not to limit its scope, which is defined by the claims. Those of ordinary skill in the art will recognize a variety of equivalent variations on the description that follows.

[0033] Figure 1 depicts a user interface view of a system for handling generic business documents exchanged among trading partners. This illustrates an environment in which aspects of the present invention may be practiced. A user of a document exchange system uses a computer, terminal or other workstation to access a login screen 111. Standard login and authentication protocols can be followed. After a successful login, a user is presented with an initial mailbox screen 112. Figure 2 depicts a sample initial screen. The user's choices from the initial screen may include folder management 121, document searching 122, listing of documents in the selected folder 123 or selection of a particular document type of interest 124. Some of these

options may be omitted or others may be added to a system. The folder management screen 121 provides typical functions for creating, deleting, renaming folders, etc. The document search screen 122 allows a user to select a document type and search on various criteria that are adapted to the selected document type. Figure 5 depicts a sample search screen. The list of documents in a folder screen 123 can be reached from either the initial screen or from a document search screen. It provides various information about documents in a folder, as shown in figure 3. The document type selection screen 124 can be implemented to allow a user to create a document from scratch. When a user selects a particular type of document to compose, the system can respond with a document compose screen 134 that includes the fields to be completed.

**[0034]** From the list of documents in a folder screen, the user can export a document 131, read it 132, or use it as a basis for a new document 133, either by replying to or copying the base document. The document may be exported as an XML, PDF, CSV, HTML or other-formatted document. Standard or user-supplied export filters can be implemented.

**[0035]** The processing of a user request to view a document can be understood by reference to the co-pending application for A Method and Apparatus for Viewing Electronic Commerce-Related Documents at pp. 5-52 and the figures cited therein. In general, a series of style sheets can be constructed for displaying documents. These style sheets may be written in XSLT, or another transformation language applicable to the data type of the documents. A series of style sheets may be written, from generic to highly customized. A rule selector can be used to select among the available style sheets based on criteria such as document type, marketplace identity, sender identity, receiver identity, portal identity or other selected criteria. A directory tree, database or other data structure can be used to access style sheets based upon the criteria used.

**[0036]** A user can select among document types for a resulting reply document 133. The available document types for a resulting reply document depend on the document type of the starting document type, for instance a starting document selected from a list of documents 123. Figure 4 is a sample screen from which a user selects a document type 442 for a resulting reply document. The reply document type can be the same as the starting type, for instance when a user is copying and reusing a

previous order as a template. The trading partner 443 may be identified by a full name, a short name and/or a unique identifier string.

**[0037]** Figure 2 provides additional details of the folder management screen. The user is presented with standard screen functionality such as home, help and log off

211. The current user and user organization are presented in the header 212. This is helpful for users who may belong to more than one organization. Options for accessing additional screens, such as the in box, composing a document, searching, addressee's and folders are presented. For figure 2, the active option is folders.

Several columns of information about folders are provided, including folder name 222, number of documents in a folder 223 and number of unread documents in the folder 224. Additional columns of information can be provided.

**[0038]** Figure 3 provides additional details of the file list screen. General aspects of the screen follow the format of other screens, e.g. 211, 212, 221. The current folder being viewed and other folders available for view can be listed in the

navigation box 331. The checkboxes 332 are provided for selecting individual documents to delete or to move to a folder. A column identifying the trading partners from whom documents originated 333 is provided. Date and time information and document types also are provided 334, 335. Icons for the document also can be provided 336. From the envelope icon, the user will know if the document has been read or not and also may know if it has been forwarded or replied to. From the export icon, the user can export the document. The same functionality can be provided for exporting a series of documents that have been checked. From the clip icon, the user can know if there are attachments associated with the document. The clip icon may allow direct access to a list of attachments or to the attachments themselves.

**[0039]** The document type selection screen of figure 4 can lead to either a search or a reply action, depending on the context. In figure 4, a trading partner name and document type may be described in the banner 441, if they have been selected.

Selection of a document type provides a context for either searching or replying.

Figure 5 depicts a sample search screen that includes many of the same features as other sample screens, e.g., 211, 212, 221, and 331. One or more field selection filters 512, 513 are supplied. The values accessible by the pull down menus of these filters are context sensitive to a selected document type, if the document type is selected by a filter 518 or has been selected by context in a prior screen. Only the field types that

are valid for a particular document type appear in the pull down menu, once a document type has been selected. One or more value specification fields 514 also are provided. These value specification fields are context sensitive to the document fields. The user's entry can be checked for format and data type as the user enters the data. The combination of field selection filters and value specification fields, context sensitive to the document type selected, enables a generic interface for searching across different document types. In addition to generic field handling, additional filters can be provided to search by date 515, document status 516, trading partner 517 or other field. A variety of interfaces can be used to implement these filters, such as pull down menus, radio buttons, checkboxes, or blanks to fill in. A list of folders to search in 519 can be dynamically generated, to match the list of folders in the navigation box 221, 331.

**[0040]** Figure 6 is a block diagram, at a very high level, depicting flow through activities related to exchange of documents between trading partners. A trading partner 601 using a connector sends a business document to a document router 606, which routes the document to a generic document handling service 609 within a service environment 608. The generic document handling service does some processing and, assuming no fatal error, stores the document inside database 607. Later, another trading partner 602 using a browser or other software adapted to communicate with the market site 603 accesses the Web server 604 running a document access servlet 605. The trading partner reads the document and, optionally, creates a response or uses the document as a basis for creating a new document. The second trading partner can access and create generic documents using a standard Web browser or other software. The servlet retrieves the starting document from the database 607. After a response or new document is created, the document can be sent to another trading partner 601. All of the functions of the market site 603 can be hosted on a single server. In some implementations, it is desirable to have separate servers for the router 606, the services 608, the database 607 and the Web servers 604. Additional servers or families of servers may be added for load balancing, high availability and/or security reasons.

**[0041]** Figure 7 is a block diagram depicting an alternative system. Interacting with the system are an entity 701 which sends a document to the system and a user 702 who interacts with the document stored by the system. The entity sends the

document to a server 708 via router and communication channels that are not depicted in the figure. The document may reach the server 708 based on the addressee of the document or a combination of addressee and document type, or it may be sent directly to a location specified by the sender. For instance, a global directory may be published that identifies locations to which participants in a marketplace desire for various types of documents to be sent. The locations may be specified in the form of market participant IDs or URLs. A market participant may have multiple IDs or URLs.

**[0042]** A document being sent is typically a self-describing, structured document.

XML documents are a common type of self-describing, structured documents. Fields within this document are self-describing, as the fields are tagged. A sample document having two different types of tagged fields is illustrated in figure 8. The document 800 may have one or more parts 830, 840. For instance, the document may include a MIME header and an XML body. A MIME header may be compliant with RFC 822. The header 830 includes a plurality of tagged fields 831-833. The body 840 also includes a plurality of tagged fields 841-849. Alternatively, the document may have only one part or one type of tagged fields. These tagged fields may comply with a schema, such as an xCBL schema by Commerce One or a CXML schema by Areba. A scheme is characterized by tagged fields having types and super types. Types inherit properties from the super types on which they are based. Types are defined based on their super types and having additional properties. Other standards to which tagged fields may comply include Sox by Commerce One, ECX, the OAGI standard of Open Applications.org, the BizTalk standard by Microsoft, the Rosetta net standard by Rosetta Net.org, and EDI X12 A32.

**[0043]** Referring to figure 7, the entity which sent a document to the system 701 may be an electronic trading partner or otherwise an electronic correspondent of the entity to which user 702 belongs. The relationship of trading partners or electronic correspondence is illustrated in figure 9. The trading partners participate in the trading network 900. This network may be hosted by a single entity or it may be a collaboration of networks hosted by distinct entities. The trading partners may both use the same host or they may use different hosts which route messages among themselves. Two trading partners 860, 880 are illustrated. Participants on behalf of the trading partners 861, 381 may be systems or human beings. Referring again to



figure 7, the entity 701 which sends the document to the server 708 may be either a system or a human being. Purchasing systems are examples of entities which generate self-describing, structured, tagged field documents and send them to systems.

- 5 **[0044]** On server 708, one or more services 711-713 may be available. One service may receive the document 711. The same or another service 712 may persistently store the incoming document, for instance in a database 707. One or more databases may be used to store data useful for electronic commerce or other document exchange. A database may include a repository of schemas 741 for
- 10 standard and entity-defined business documents, a repository of JavaBeans 742, C++ structures, Pascal records or scripts useful to electronic commerce, a document map repository 743 for translation of documents from one format to another (e.g., the xCBL format to an export format) and for transformation of documents from one type to another (e.g., from a request for quotation to a quotation.) The database may further
- 15 include a report layout repository 744 and a presentation layout repository 745. The presentation layout repository may include declarative transformations for changing documents from a Sox document format to an HTML format, and back again. The transmission properties data describes the transport information needed to sent a business document to a recipient. For example, SSL security credentials can be stored
- 20 as transmission properties. A trading partner directory is also useful. The trading partner directory may, as described above, identify URLs to which the sending entity 701 transmits documents. The persistent storage for data 707, 741-47 may be on a single data storage unit or multiple units. Persistent document storage need not be part of a database. An indexed flat file would suffice to store XML-compliant
- 25 documents. The services host 708 may also host an indexing service to index one or more of the tagged fields. Alternatively, a database system managing the persistent storage or other subsystem may index documents for retrieval. An incoming document may or may not be validated against schema from a schema repository 741, before it is persistently stored. A schema may be used to interpret the document. One
- 30 or more JavaBeans or Scripps may be used to act on the document before it is stored.
- [0045]** One schema for persistently stored documents is illustrated in figure 10. In this schema, a document and its header are captured in structure 1020. If this document is a copy or a reply based on an original document, the original document is

stored in structure 1023. A copy or a reply document is associated with a special folder 1024. Document attachments are captured in structure 1040. Sets of valid document types and document statuses are maintained in structures 1010 and 1030, respectively. The one-to-many connections between structures 1010, 1030 and structure 1020 indicate that only one document type and one document status are allowed per document. To support searching, searchable fields are listed in a data structure 1012, and may be assigned unique or non-unique alias names. Information regarding searchable fields is listed in a separate data structure 1022. Other data arrangements may equally well practice aspects of the present invention.

**[0046]** Persistent storage of the incoming document may be accompanied by various processing steps. The original document, prior to normalizing, may be stored in a database. Envelope properties may be extracted from the envelope and normalized in a database or other storage. A predefined list of indexed fields, by document type, may be consulted and those fields indexed. Generic document properties, such as date and status, may be extracted from the envelope. The envelope itself may be separately stored. Attachments may be separately stored. Unneeded white space may be removed and name space abbreviations used to reduce storage requirements.

**[0047]** Returning to figure 7, the service 713 may advise the user 702 of receipt of the document by messaging, posting any other practical means. One or more users may be given notice, based on the identity of the sending entity, the identity of the receiving entity, the document type or other characteristics of the document. One syntax for subscription by a user is subscription = sender ID.recipient ID.document type. To receive all Order documents from a particular sending entity, a user could subscribe as follows: subscription = S1234.\*.Order. This subscription would cause service 713 to notify user 702 of receipt of order type documents from sending entity S1234. Security features of the system would, of course, restrict access appropriately. Notice may be given by messaging, such as e-mail or Lotus Notes messaging. A message may include a subject, such as, "five new documents received," header text generally stating that new documents have been received or that old documents remain to be viewed. The e-mail may further include body text providing the date, number and sending party's identities for documents received. The body text may further provide detail regarding individual documents received, such as the document

type, sender identity and date or time of receipt. The user interface may identify the location for viewing the document, such as by a click through URL. Alternatively, notice may be given by posting at a location to which the user has access. A combination of notice formats may be used, such as posting followed by e-mail

- 5 follow-up in the user does not promptly access listed documents. E-mails based on the status of accessing a document or a post notice may be sent periodically and may include increasingly strong wording or additional addresses, based on the type and/or aging of the received document.

**[0048]** A system 704 may be based upon a web server and a servlet container, for  
 10 instance, compatible with JRun 3. A user interface application 721 may include a homepage 721, an in box 722, one or more services to read the document and/or its attachments 723, services for folder management 724, customized folders 725, searching for and listing documents. An additional service may provide access for downloading template documents 728. The web server may include Microsoft's  
 15 WebServer software, a Java interpreter such as JRun 3 and a servlet container. A data storage interface may use the resources of a database. A first collection of services 720 is illustrated as being coupled with a transformation engine 730. The transformation engine may include selection logic 732 to select the style sheet for transformation purposes and transformation logic 731 which applies a style sheet to a  
 20 document.

**[0049]** Figure 11 depicts a further alternative environment in which aspects of the present invention can be practiced, similar to the environments depicted in figures 6 and 7. A web server 1111 takes HTTP requests and makes HTTP responses. A generic document handling facility takes advantage of the web server resources  
 25 through a program interface. One useful API is Sun Microsystems Java servlet. The so-called servlet technology provides a simple, consistent mechanism for providing access to web server services. A servlet written in Java can provide a component-based, platform-independent method for building Web-based applications. A single servlet application may be integrated with a variety of web-enabled application  
 30 servers such as BEA WebLogic Application Server, IBM WebSphere, iPlanet Application Server others. Complementary to servlet technology is Java Server Pages (JSP) support 1114. The JSP technology is an extension of servlet technology that supports authoring of HTML and XML pages. JSP facilitates mixing dynamic

content with static templates. JSP supports encapsulation of logic to generate content dynamically. The dynamic content logic can be accessed from the template page.

The servlet runner 1112 resident on the web server may be JRun or a similar Java environment. Alternatively, a different program interface to the web server may be used with a different programming language. The servlet runner supports the servlet interface for access to the web server. The servlet runner supports the servlet controller 1113, invocation of JSP pages 1114, and a variety of applications 1121-26. Among these applications, the data access layer is an abstraction layer for access to the database 1151. The data access layer is present on each server or in each

application environment that accesses the database 1133, 1144. XMLPres 1122 is a style sheet engine that handles extended XSLT style sheets for transformation of XML into HTML. The rule selector 1123, as described in the co-pending application, provides access to default or a customized rules and templates that match parameter such as document type and trading partner. The XML engine parses and processes XML documents. For XML, records can be parsed and processed using DOM, SAX or any other programming model. Folder management is provided 1125. Software and resources for processing browser forms with embedded pass specifications are also provided 1126, as described further below. E-mail notification 1131 may be structured with an e-mail module 1132, a data access layer 1133, and e-mail management routines. The e-mail module will utilize routines of the data access layer and the e-mail management library. Sun Microsystems provides a library of e-mail management routines complementary to its servlet technology. The overall generic document exchange service 1141 can be hosted within a services environment 1142. The generic document exchange service 1143 may include logic in resources for data access 1144, XML transformation is 1145 and XML searching and processing 1146. Data accessed and utilized by these routines may be stored in multiple databases 1151, 1152. A database of documents may be one component and market site data may be another, the market site data including a directory compliant with a lightweight directory access protocol.

**[0050]** Figure 12 is a block diagram of a user interface and declarative data. This figure provides additional detail beyond the detail of figure 1. The initial three screens, for displaying a document 132, selecting a document type 133, and composing a reply or new document 141 are the same as in figure 1. Figure 12

depicts the use of declarative data for a mapping selector 1231, for data transformation mapping 1211, and for display transformation 1212 in conjunction with the document type selection screen 133. The mapping selector 1231 provides a list of document types that can be produced from a selected starting document is

5 accessed and incorporated in the user interface. The types of resulting documents that can be produced may depend on the marketplace, trading partner and other factors, as well as the document type of the starting document. From a starting document and selection of a document type for a resulting document, data transformation mapping 1211 is applied to generate a draft resulting document from the starting document.

10 The transformation mapping should ensure that the resulting document is well formed, when the starting document is well formed. For XML to XML transformation, the declarative transformation mapping may take the form of an XSLT style sheet, potentially with extensions to XSLT. Other forms of declarative schema-to-schema mapping may also be used. Display transformation mapping

15 utilizes declarative display format data 1121 to transform the underlying data for use in the user interface, for instance, to transform XML to HTML they can be viewed by a browser. Following document type selection 133, application of declarative data from a mapping selector, data transformation mapping, and display transformation mapping is used to produce a document updates screen 141. In one embodiment,

20 application of one or more declarative transformations to the starting document produces a user interface form. The user interface form includes a plurality of path specifications for fields corresponding to the document type of the resulting document, starting values based on the starting document for at least some of the fields in the resulting document, and values to be completed for other fields in the

25 resulting document. Field labels also may be included, either as data or along with other page formatting information. Alternatively, all of the fields may be populated and none left to be completed. Some fields may be editable and others not editable. The path specifications may conveniently be implemented as HTML hidden fields or otherwise as non-displaying fields. They can be displayed, for instance, for

30 debugging. XPath is one convention that can be used for path specifications to nodes or fields within an XML document. An alias for an XPath path specification also may be used, in conjunction with a lookup from the alias to the full path specification. Aliases can double as field labels.

[0051] Sample excerpts of a document to display transformation 1212 for a main processing routine and display of header information with embedded path specifications follows:

#### Default.xsl

```

5      <?xml version="1.0" ?>
- <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ext="http://www.commerceone.com/xmlpres" xmlns:cbl="urn:x-
10  commerceone:
  document:com:commerceone:XCBL30:XCBL30.sox$1.0"
  xmlns:xm="http://www.commerceone.com/xcbllmailbox">
  <xsl:include href="schematronerror.xsl" />
  <xsl:include href="header_display.xsl" />
15  <xsl:include href="detail_display.xsl" />
  <xsl:include href="summary_display.xsl" />
  <xsl:include href="lr:Quote:Quote_variable.xsl$1.0" />
  <xsl:decimal-format name="DEIT" decimal-separator="," grouping-
    separator="." />
20  <xsl:decimal-format name="FR" decimal-separator="," grouping-
    separator=""/> />
- <xsl:template match="/">
  - <!--
    This is for displaying Schematron errors
25  -->
  - <xsl:if test="string($xm:hasSchematron)='true'">
    <xsl:variable name="results" select="document('error:///
      schematronConformance.xml')"/>
    - <xsl:call-template name="ErrorDisplay">
      <xsl:with-param name="confdoc" select="$results" />
      <xsl:with-param name="idmap" select="$schematronmaps" />
30    </xsl:call-template>
    </xsl:if>
    - <!--
      If the xml instance has error, display error fields with
      field validation error
35  -->
    - <xsl:choose>
      - <xsl:when test="string($xm:hasError)='true'">
40        <xsl:variable name="ErrorDocument" select="document('error:/
          //inMemoryError')"/>
        - <xsl:variable name="Prefixes">
          <empty />
        </xsl:variable>
45        - <xsl:call-template name="DisplayHeader_Error">
          <xsl:with-param name="Prefixes" select="$Prefixes" />
          <xsl:with-param name="ErrorDocument"
            select="$ErrorDocument" />

```

```

        <xsl:with-param name="ErrorCodeMaps" select="$codemaps"
        />
    </xsl:call-template>
    - <xsl:call-template name="DisplayDetail_Error">
5      <xsl:with-param name="Prefixes" select="$Prefixes" />
      <xsl:with-param name="ErrorDocument"
        select="$ErrorDocument" />
      <xsl:with-param name="ErrorCodeMaps" select="$codemaps"
        />
10    </xsl:call-template>
    <xsl:call-template name="DisplaySummary" />
  </xsl:when>
  - <!--
    If the xml instance doesn't contain error, display all
15    fields without any error
    -->
  - <xsl:otherwise>
    <xsl:call-template name="DisplayHeader_NoError" />
    <xsl:call-template name="DisplayDetail_NoError" />
20    <xsl:call-template name="DisplaySummary" />
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

### Header display.xsl

```

<?xml version="1.0" encoding="UTF-8" ?>
- <xsl:stylesheet version="1.0"
30   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:ext="http://www.commerceone.com/xmlpres" xmlns:cbl="urn:x-
   commerceone:
   document:com:commerceone:XCBL30:XCBL30.sox$1.0"
   xmlns:xm="http://www.commerceone.com/xcbllmailbox">
35   <xsl:include href="error.xsl" />
   <xsl:include href="header_party_display.xsl" />
  - <xsl:template name="DisplayHeader_Error">
    - <!--
      This is the prefix to namespace bindings used in the
40      location
    -->
    <xsl:param name="Prefixes" />
  - <!--
    This is the error document to retrieve errors from
45    -->
    <xsl:param name="ErrorDocument" />
    - <!--
      This is a result tree fragment that maps error codes
      to localized descriptions

```

```

-->
<xsl:param name="ErrorCodeMaps" />
- <TABLE cellpadding="1" cellspacing="0" border="0" width="100%">
  - <TR class="ListHeader">
    - <TD align="left" colspan="2" class="ListHeaderText">
      <xsl:value-of select="$QuoteHeader" />
    </TD>
  </TR>
  - <TR>
    - <TD class="StatusHeaderSmallNew" width="50%">
      <span class="RequiredField">*</span>
      <xsl:value-of select="$QuoteRefNum" />
    </TD>
    - <TD class="StatusHeaderSmallNew" width="50%">
      <span class="RequiredField">*</span>
      <xsl:value-of select="$QuoteIssueDate" />
      (
        <xsl:value-of select="string($xm:DATETIMEMASK)" />
      )
    </TD>
  </TR>
  - <TR>
    - <TD class="ElementStyle" width="50%">
      <xsl:variable name="xpath">/Quote/QuoteHeader/QuoteID/
        Reference/RefNum</xsl:variable>
      <input type="hidden" name="{concat('XPostChange:', $xpath)}"
        value="" />
      <input class="ElementStyle" type="text"
        name="{concat('XPostContent:', $xpath)}" value="{cbl:Quote/
        cbl:QuoteHeader/cbl:QuoteID/cbl:Reference/cbl:RefNum}"
        size="20"
        onChange="javascript:setModified('{concat('XPostChange:',
        $xpath)}');" />
    - <xsl:variable name="RefNumErrors">
      - <xsl:call-template name="ErrorDetector">
        <xsl:with-param name="Location"
          select="/Quote/QuoteHeader/QuoteID/Reference/RefNum"
        />
        <xsl:with-param name="Prefixes" select="$Prefixes" />
        <xsl:with-param name="ErrorDocument"
          select="$ErrorDocument" />
        <xsl:with-param name="ErrorCodeMaps"
          select="$ErrorCodeMaps" />
      </xsl:call-template>
    </xsl:variable>
    - <xsl:if
      test="string($RefNumErrors)!=string($NotMappedConstant)"
      >
      - <span class="ErrorTextSmall">

```



```

[
  <xsl:value-of select="$Error" />
  <xsl:value-of select="$RefNumErrors" />
]:
5   </span>
  </xsl:if>
</TD>
- <TD class="ElementStyle" width="50%">
  <xsl:variable name="xpath">/Quote/QuoteHeader/
10   QuoteIssueDate</xsl:variable>
  <input type="hidden" name="{concat('XPostChange:', $xpath)}"
    value="" />
  - <xsl:variable name="DateErrors">
    - <xsl:call-template name="ErrorDetector">
      <xsl:with-param name="Location" select="/Quote/
15       QuoteHeader/QuoteIssueDate" />
      <xsl:with-param name="Prefixes" select="$Prefixes" />
      <xsl:with-param name="ErrorDocument"
        select="$ErrorDocument" />
      <xsl:with-param name="ErrorCodeMaps"
        select="$ErrorCodeMaps" />
    </xsl:call-template>
    </xsl:variable>
    <xsl:param name="date" select="cbl:Quote/
25     cbl:QuoteHeader/cbl:QuoteIssueDate" />
  - <xsl:param name="QuoteIssuedDate">
    <ce:lookup xmlns:ce="http://www.commerceone.com/xslt/
      extensions" xmlns:mailllookup="urn:mailbox:lookup"
      mapname="mailllookup:DateTimeFormatter" value="$date"
30     errorvalue="Can not mapped" />
    </xsl:param>
  - <xsl:choose>
    - <xsl:when
      test="string($DateErrors)!=string($NotMappedConstant)">
35     <input class="ElementStyle" type="text"
      name="{concat('XPostContent:', $xpath)}"
      value="{cbl:Quote/
        cbl:QuoteHeader/cbl:QuoteIssueDate}" size="15"
      onChange="javascript:setModified('{concat('XPostChang
40     e:', $xpath)}');" />
    - <span class="ErrorTextSmall">
      [
        <xsl:value-of select="$Error" />
        <xsl:value-of select="$DateErrors" />
45      ]:
      </span>
    </xsl:when>
  - <xsl:otherwise>

```

<input class="ElementStyle" type="text"  
   name="{concat('XPostContent:', \$xpath)}"  
   value="{ \$QuoteIssuedDate}" size="15"  
   onChange="javascript:setModified('{concat('XPostChange:', \$xpath)}');" />  
 </xsl:otherwise>  
 </xsl:choose>  
 </TD>  
 </TR>  
 <xsl:call-template name="DisplayParty" />  
 </TABLE>  
   
 <BR />  
 </xsl:template>  
 - <xsl:template name="DisplayHeader\_NoError">  
   - <TABLE cellpadding="1" cellspacing="0" border="0" width="100%">  
     - <TR class="ListHeader">  
       - <TD align="left" colspan="2" class="ListHeaderText">  
         <xsl:value-of select="\$QuoteHeader" />  
       </TD>  
     </TR>  
     - <TR>  
       - <TD class="StatusHeaderSmallNew" width="50%">  
         <span class="RequiredField">\*</span>  
         <xsl:value-of select="\$QuoteRefNum" />  
       </TD>  
       - <TD class="StatusHeaderSmallNew" width="50%">  
         <span class="RequiredField">\*</span>  
         <xsl:value-of select="\$QuoteIssuedDate" />  
         (  
           <xsl:value-of select="string(\$xm:DATETIMEMASK)" />  
         )  
       </TD>  
     </TR>  
     - <TR>  
       - <TD class="ElementStyle" width="50%">  
         <xsl:variable name="xpath">/Quote/QuoteHeader/QuoteID/  
           Reference/RefNum</xsl:variable>  
         <input type="hidden" name="{concat('XPostChange:', \$xpath)}"  
           value="" />  
         <input class="ElementStyle" type="text"  
           name="{concat('XPostContent:', \$xpath)}" value="{cbl:Quote/  
           cbl:QuoteHeader/cbl:QuoteID/cbl:Reference/cbl:RefNum}"  
           size="20"  
           onChange="javascript:setModified('{concat('XPostChange:',  
           \$xpath)}');" />  
       </TD>  
       - <TD class="ElementStyle" width="50%">

```

<xsl:param name="date" select="cbl:Quote/
  cbl:QuoteHeader/cbl:QuoteIssueDate" />
<xsl:variable name="xpath">/Quote/QuoteHeader/
  QuoteIssueDate</xsl:variable>
5  <input type="hidden" name="{concat('XPostChange:', $xpath)}"
   value="" />
- <xsl:param name="QuoteIssuedDate">
  <ce:lookup xmlns:ce="http://www.commerceone.com/xsl/
    extensions" xmlns:maillookup="urn:mailbox:lookup"
10  mapname="maillookup:DateTimeFormatter" value="$date"
    errorvalue="Can not mapped" />
  </xsl:param>
  <input class="ElementStyle" type="text"
15  name="{concat('XPostContent:', $xpath)}"
    value="{ $QuoteIssuedDate}" size="15"
    onChange="javascript:setModified('{concat('XPostChange:',
      $xpath)}');" />
  </TD>
</TR>
20 <xsl:call-template name="DisplayParty" />
  
  <BR />
</TABLE>
</xsl:template>
25 </xsl:stylesheet>

```

[0052] A draft resulting document also may be generated, in the process of producing the user form. This document may be produced according to a declarative transformation of a starting XML document into a draft resulting XML document. The draft resulting document may include starting values from the starting document. It also may include default values for some fields and directions for completing other fields. The draft resulting document may be stored in memory according to a document-object-model (DOM) or another tree-based representation. Alternatively, the draft resulting document may be stored on disk or in memory in a form compatible with Simple API for XML (SAX or SAX2) or another event-based access model. Or, an event-based API can be used to construct a tree, or traverse an in-memory tree. When a draft resulting document is generated, in addition to a form, the draft resulting document may be maintained in memory while the user works from the form and a document ID can be maintained with other state information for the draft resulting document. Alternatively, a document ID and related information can be transmitted with the user form, even in a stateless fashion, and the draft resulting

document constructed after the user's updating of the form. A draft resulting document can be transformed with a display transformation to generate the user form. The display transformation may be generated with an XSLT stylesheet or another set of declarative data.

- 5 [0053] Sample excerpts of a document to document transformation 1211 from a request for quotation to quotation follows, including portions of a main processing routine and a party copy routine:

**Default.xsl**

```

10 - <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ext="http://www.commerceone.com/xmlpres" xmlns:cbl="urn:x-
    commerceone: document:com:commerceone:XCBL30:XCBL30.soxx$1.0"
    xmlns:xm="http://www.commerceone.com/xcbllmailbox">
    <xsl:output method="xml" indent="yes" omit-xmldeclaration="yes" />
15 <xsl:include href="reference_copy.xsl" />
    <xsl:include href="party_copy.xsl" />
    <xsl:include href="lineitemnum_copy.xsl" />
    <xsl:include href="itemidentifiers_copy.xsl" />
    <xsl:include href="identifier_copy.xsl" />
20 <xsl:include href="totalquantity_copy.xsl" />
    - <xsl:template match="/">
        - <xsl:processing-instruction name="soxtype">
            <xsl:text>urn:xcommerceone:
                document:com:commerceone:XCBL30:XCBL30.soxx$1.0</xsl:text>
25 </xsl:processing-instruction>
        - <Quote>
            <xsl:apply-templates select="cbl:RequestForQuotation/
                cbl:RequestQuoteHeader" />
            <xsl:apply-templates select="cbl:RequestForQuotation/
30 cbl:ListOfRequestQuoteDetails" />
            <xsl:apply-templates select="cbl:RequestForQuotation/
                cbl:RequestQuoteSummary" />
            </Quote>
            </xsl:template>
35 - <xsl:template match="cbl:RequestQuoteHeader">
            <xsl:call-template name="OutputQuoteHeader" />
            </xsl:template>
            - <xsl:template match="cbl:ListOfRequestQuoteDetails">
            <xsl:call-template name="OutputListOfQuoteDetails" />
40 </xsl:template>
            - <xsl:template match="cbl:RequestQuoteSummary">
            <xsl:call-template name="OutputQuoteSummary" />
            </xsl:template>
            - <xsl:template name="OutputQuoteHeader">
45 - <QuoteHeader>
            <QuoteIssueDate />

```

```

- <QuoteID>
  <xsl:apply-templates select="cbl:RequestQuoteID" />
</QuoteID>
- <QuoteParty>
  <xsl:apply-templates select="cbl:RequestQuoteParty" />
</QuoteParty>
  <xsl:call-template name="OutputQuoteCurrency" />
  <xsl:call-template name="OutputQuoteTermsOfPayment" />
</QuoteHeader>
</xsl:template>
- <xsl:template name="OutputListOfQuoteDetails">
  - <ListOfQuoteDetails>
    - <xsl:for-each select="cbl:RequestQuoteDetails">
      - <QuoteDetails>
        <xsl:call-template name="OutputQuoteItemtype" />
        <xsl:call-template name="OutputQuoteItemDetail" />
      </QuoteDetails>
    </xsl:for-each>
  </ListOfQuoteDetails>
</xsl:template>
- <xsl:template name="OutputQuoteSummary">
  - <QuoteSummary>
    <xsl:copy-of select="cbl:TotalNumberOfLineItems" />
  </QuoteSummary>
</xsl:template>
- <xsl:template name="OutputQuoteItemtype">
  - <QuoteItemtype>
    - <QuoteType>
      <QuoteTypeCoded />
    </QuoteType>
  </QuoteItemtype>
</xsl:template>
- <xsl:template name="OutputQuoteItemDetail">
  - <QuoteItemDetail>
    <xsl:apply-templates
      select="cbl:RequestQuoteItemDetail/cbl:BaseItemDetail" />
    </QuoteItemDetail>
  - <QuotePricingDetail>
    - <PricingDetail>
      - <ListOfPrice>
        - <Price>
          - <UnitPrice>
            <UnitPriceValue />
            - <UnitOfMeasurement>
              <UOMCoded />
            </UnitOfMeasurement>
          </UnitPrice>
        </Price>
      </ListOfPrice>
    </PricingDetail>
  </QuotePricingDetail>
</xsl:template>

```

```

- <TotalValue>
  - <MonetaryValue>
    <MonetaryAmount />
    </MonetaryValue>
  </TotalValue>
</PricingDetail>
</QuotePricingDetail>
</xsl:template>
- <xsl:template match="cbl:BaselItemDetail">
  <xsl:call-template name="CopyLineItemNum" />
  <xsl:call-template name="CopyItemIdentifiers" />
  <xsl:call-template name="CopyTotalQuantity" />
</xsl:template>
- <xsl:template name="OutputQuoteCurrency">
  - <xsl:if test="cbl:RequestQuoteCurrency/cbl:Currency/
    cbl:CurrencyCoded">
    - <QuoteCurrency>
      <xsl:copy-of select="cbl:RequestQuoteCurrency/ cbl:Currency" />
    </QuoteCurrency>
    </xsl:if>
  </xsl:template>
- <xsl:template name="OutputQuoteTermsOfPayment">
  - <xsl:if test="cbl:RequestQuoteTermsOfPayment">
    - <QuoteTermsOfPayment>
      - <PaymentInstructions>
        <xsl:call-template name="OutputQuotePaymentTerms" />
        <xsl:call-template name="OutputQuotePaymentMethod" />
      </PaymentInstructions>
    </QuoteTermsOfPayment>
    </xsl:if>
  </xsl:template>
- <xsl:template name="OutputQuotePaymentTerms">
  - <xsl:for-each
    select="cbl:RequestQuoteTermsOfPayment/cbl:PaymentInstructions/
    cbl:PaymentTerms">
    - <PaymentTerms>
      - <xsl:for-each select="cbl:PaymentTerm">
        - <PaymentTerm>
          <xsl:copy-of select="cbl:PaymentTermCoded" />
          <xsl:copy-of select="cbl:PaymentTermCodedOther" />
          <xsl:copy-of select="cbl:PaymentTermValue" />
          <xsl:copy-of select="cbl:PaymentTermDetails" />
        </PaymentTerm>
      </xsl:for-each>
    </PaymentTerms>
  </xsl:for-each>
</xsl:template>
- <xsl:template name="OutputQuotePaymentMethod">

```

```

- <xsl:for-each select="cbl:RequestQuoteTermsOfPayment/
  cbl:PaymentInstructions/cbl:PaymentMethod">
  - <PaymentMethod>
    <xsl:copy-of select="cbl:PaymentMeanCoded" />
    <xsl:copy-of select="cbl:PaymentMeanCodedOther" />
    <xsl:copy-of select="cbl:PaymentMeanReference" />
    <xsl:copy-of select="cbl:PaymentSystemCoded" />
    <xsl:copy-of select="cbl:PaymentSystemCodedOther" />
    <xsl:copy-of select="cbl:OriginatingFIAccount" />
    <xsl:copy-of select="cbl:ReceivingFIAccount" />
    <xsl:copy-of select="cbl:CardInfo" />
  </PaymentMethod>
</xsl:for-each>
</xsl:template>
- <xsl:template match="cbl:RequestQuoteID">
  <xsl:call-template name="CopyReference" />
</xsl:template>
- <xsl:template match="cbl:RequestQuoteParty">
  - <OrderParty>
    - <BuyerParty>
      <xsl:apply-templates select="cbl:OrderParty/
        cbl:BuyerParty/cbl:Party" />
    </BuyerParty>
    - <SellerParty>
      <xsl:apply-templates select="cbl:OrderParty/
        cbl:SellerParty/cbl:Party" />
    </SellerParty>
  </OrderParty>
</xsl:template>
- <xsl:template match="cbl:Party">
  <xsl:call-template name="CopyParty" />
</xsl:template>
</xsl:stylesheet>

```

### 35 Party\_copy.xsl

```

- <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ext="http://www.commerceone.com/xmlpres" xmlns:cbl="urn:x-
  commerceone: document:com:commerceone:XCBL30:XCBL30.soxx$1.0"
  xmlns:xm="http://www.commerceone.com/xcbllmailbox">
  - <xsl:template name="CopyParty">
    - <Party>
      <xsl:apply-templates select="cbl:PartyID" />
      <xsl:call-template name="CopyNameAddress" />
    </Party>
  </xsl:template>
  - <xsl:template match="cbl:PartyID">
    - <PartyID>

```

```

        <xsl:apply-templates select="cbl:Identifier" />
    </PartyID>
</xsl:template>
- <xsl:template name="CopyNameAddress">
5   <xsl:copy-of select="cbl:NameAddress" />
    </xsl:template>
</xsl:stylesheet>

```

[0054] The user acts upon the form 141. Fields are added, completed or changed.

10 More than one iteration may be required, for instance, if a user adds a line item to a quotation. When the user is ready, the form is posted to the server for validation. One or more types of validation can be applied, such as field validation against a schema (e.g., for well-formedness and valid ranges) and business processing validation against a rule base. For instance, a SOX-compliant xCBL schema and a set

15 of Schematron rules can be used for validation. In Figure 12, the flow branches from 141 depending on whether field validation is error free. An error free field validation is displayed 1224 with a message. A document containing field errors results in generation of an error document that is merged for display with the original document by a declarative transformation 1212, thereby displaying validation errors in the same

20 user interface as the document being edited 1213. The user's actions may be iterative, to correct one or more errors before repeating field validation. The user may be presented 1225 with further options of validating the document against business processing rules applicable to one or more trading partners or of saving the document as a draft 1233. Alternatively, the multiple types of validations may be performed on

25 the same document, without repeated user requests for validation. In one embodiment, business processing rules are applied by a Schematron engine 1232.

[0055] Excerpts from a sample of a user form 141 that produced the sample screen in Figure 13 follow:

```

30 <!-- ----- start Mailbox20.jsp ----- -->

    <HTML>
    <HEAD>
    <TITLE>xCBL Mailbox 2.0</TITLE>
35   <LINK href='../css/main.css' rel=STYLESHEET type=text/css>
    </HEAD>
    <BODY bgColor=#ffffff leftMargin=0 rightMargin=0 topMargin=0
    MARGINWIDTH="0" MARGINHEIGHT="0">

```



```

<script language="javascript" src="../waiting.js">
</script>
<script language="javascript" src="../displayhelp.js">
</script>

```

```

<!-- ----- start logo.jsp ----- -->

```

```

<TABLE border=0 cellPadding=0 cellSpacing=0 width="100%">
<TR>
  <TD class=navigation colSpan=3 height=2><IMG height=1
    src='../img/spacer.gif' width=1></TD></TR>
<TR>
  <TD width="27%"><IMG height=52 src='../img/hleftlogo.gif'></TD>
  <TD noWrap width="45%">

```

\*\*\*

```

</script>
<!-- ----- end logo.jsp ----- -->

```

```

<TABLE border=0 cellPadding=0 cellSpacing=0 width="100%">
<TR class=topcell>
  <TD height=18><IMG height=1 src='../img/spacer.gif' width=1>&nbsp;</TD>
  <TD height=18 align="right" class="HomeLink">
    <A href="XMServlet?webaction=Init" class="HomeLink">
      xCBL Mailbox</A>&nbsp;&nbsp;</TD></TR>
</TABLE>
<TABLE align=center border=0 cellPadding=2 width="100%"><TBODY>
<TR>
  <TD align=middle bgColor=#e9ecef vAlign=top width="20%">

```

```

<!-- ----- start left_nav.jsp ----- -->

```

```

<TABLE border=0 cellPadding=2 cellSpacing=0 width="90%"><TBODY>
<TR>
  <TD align=middle bgColor=#000000>
    <TABLE border=0 cellPadding=0 cellSpacing=0 width="100%"
      class="LeftNav">
    <TR>
      <TD colSpan=2><IMG border=0 height=3 src='../img/spacer.gif'
        width=1></TD></TR>
    <TR>
      <TD align=middle>&nbsp;</TD>
      <TD align=left class="BoldText">
        <A
          href="XMServlet?webaction=FdrActionGetDocList&Folder_go_folder=1">
            View Inbox</A></TD></TR>

```

\*\*\*

5

10

15

\*\*\*

20

25

30

35

10

15

Page 26 of 51

```

document.folder_index.action="XMServlet?webaction=FdrActionEmptyTrash";
    document.folder_index.Folder_go_folder.value = "4";
    document.folder_index.submit();
5      }
    }
    //-->
    </script>

10      <TR>
        <TD colSpan=2><IMG border=0 height=3 src='../img/spacer.gif'
width=1></TD></TR>
        </TABLE></TD></TR>
    </TABLE>
15    <!------- end left_nav.jsp ----->

        </TD>
        <TD class=DisplayArea vAlign=top width="80%">

20          <!-- Begin MAIN CONTENT CELL -->

        <FORM ACTION="XMServlet?webaction=XMPrintAction" name="printdoc"
25      METHOD="post" target="_blank">
        <input type="hidden" name="Docid"
value="XVeETjJ6Y7lvTgK+MgDgZU0jgWzcmOux4pThkFH0oPQ=">
        </FORM>

30      <!-- Begin FORM -->
        <FORM ACTION="XMServlet" name="mainform" METHOD="post">

        <input type="Hidden" name="BackType" value ="1">
        <input type="Hidden" name="Folder_requested_page" value="2">

35          <input type="Hidden" name="Folder_go_folder" value="1">
          <input type="Hidden" name="Doclist.sort_by" value="1D">

        <input type="Hidden" name="BackToDocument" value="1">
40      <script language="JavaScript">
closeWaitingWindow('process');
        <!--
function doXPostProcess(webaction, xpost, xpath, val)
{
45      document.mainform.webaction.value=webaction;
      var fieldName = xpost + xpath;
      document.mainform[fieldName].value = val;
      document.mainform.submit();
}

```

```

function doRemoveItemLevelAttachment(webaction, xpost, xpath, val, attachuri)
{
    document.mainform.webaction.value=webaction;
5   var fieldName = xpost + xpath;
    document.mainform[fieldName].value = val;
    document.mainform.AttUri.value=attachuri;
    document.mainform.submit();
}

10  function doRemoveLineItem(webaction, xpost, xpath, val, lineitem)
    {
        document.mainform.webaction.value=webaction;
        var fieldName = xpost + xpath;
15     document.mainform[fieldName].value = val;
        document.mainform.RemoveLineItem.value=lineitem;
        document.mainform.submit();
    }

20  function doXPostProcessWithDefault(webaction, xpost, xpath, val,
    def_fieldName, def_val)
    {
        document.mainform.webaction.value=webaction;
        var fieldName = xpost + xpath;
25     document.mainform[fieldName].value = val;
        var DTnow = new Date();
        document.mainform[def_fieldName].value = def_val + ":" + DTnow.getTime();
        document.mainform.submit();
    }

30  var isModified = false;
    function setModified(xpost) {
        isModified=true;
        document.mainform[xpost].value = 'true';
35  }

    function addNewAttachment(webaction, attachmentURI, xpath_val) {
        document.mainform.webaction.value=webaction;
        document.mainform.doclevelatt.value = "1";
40     attachmentURI_fld = eval("document.mainform." + "XMAttachmentUri");
        attachmentURI_fld.value = attachmentURI;
        fileNameXPath_fld = eval("document.mainform." + "FileNameXPath");
        fileNameXPath_fld.value = xpath_val;
    }

45

    function backToDocList()
    {
        if (document.mainform.BackType.value == "1")

```

```

        document.mainform.action = "XMServlet?webaction=FdrActionGetDocList";
    else if (document.mainform.BackType.value == "2")
        document.mainform.action = "XMServlet?webaction=XMSearchDoc";
    else
5      document.mainform.action =
        "XMServlet?webaction=FdrActionRepliedDoc";
        document.mainform.submit();
    }

```

```

10  function editNotes()
    {
        if (isModified)
        {
15         if (!confirm("You have not save your changes, continue? "))
            return;
        }
        document.mainform.webaction.value="RREditNotesAction";
        document.mainform.submit();
20    }

```

```

function editAttachment()
{
    document.mainform.webaction.value="CRAttachAction";
25    document.mainform.doclevelatt.value = "0";
    document.mainform.submit();
}
// -->
</script>

```

```

30  <table width="100%" border="0" cellspacing="0" cellpadding="1">
    <tr class="Header">
        <td class="HeaderText" align="left">Draft</td>

```

```

35    <td class="HeaderText" align="right"><a href=""
onclick="javascript:backToDocList(); return false;">Back to list of
documents</a>&nbsp;&nbsp;&nbsp;</td>

```

```

    </tr>
40  </table>
    <br>

```

```

    <input type="hidden" name="webaction" value="">
    <input type="hidden" name="doclevelatt" value="0">
45  <input type="hidden" name="XPostDocumentID"
value="XVeETjJ6Y7lvTgK+MgDgZU0jgWzcmOux4pThkFH0oPQ=">
    <input type="hidden" name="Docid"
value="XVeETjJ6Y7lvTgK+MgDgZU0jgWzcmOux4pThkFH0oPQ=">
    <input type="hidden" name="XMAAttachmentUri" value="">

```

```
5 <!-- HEADER BAR -->
```

10

30

35

```

    <a href="" onclick="javascript:editAttachment(); return false;">Add/Edit
Attachments</a></td>
</tr>
</table>

```

5

```

<br/>
<table width="100%" border="0" cellspacing="0" cellpadding="0">

```

10

```

<tr>
<!-- MAIN document area -->
<td colspan="2">
<Frame>
<?xml version="1.0" encoding="UTF-8"?>

```

15

```

<TABLE xmlns:cbl="urn:x-
commerceone:document:com:commerceone:XCBL30:XCBL30:sox$1.0"
xmlns:xml="http://www.commerceone.com/xcbllmailbox"
xmlns:ext="http://www.commerceone.com/xmlpres" width="100%" border="0"
cellspacing="0" cellpadding="1"><TR class="ListHeader"><TD
class="ListHeaderText" colspan="2" align="left">Quote
Header:</TD></TR><TR><TD width="50%"
class="StatusHeaderSmallNew"><span
class="RequiredField">*</span>Quote#:</TD><TD width="50%"
class="StatusHeaderSmallNew"><span class="RequiredField">*</span>Quote
IssueDate:

```

25

```

(Please Input as MM/dd/yyyy)
</TD></TR><TR><TD width="50%" class="ElementStyle"><input value=""
name="XPostChange:/Quote/QuoteHeader/QuoteID/Reference/RefNum"
type="hidden"/><input
onChange="javascript:setModified('XPostChange:/Quote/QuoteHeader/QuoteID/
Reference/RefNum');" size="20" value="PC-009"
name="XPostContent:/Quote/QuoteHeader/QuoteID/Reference/RefNum"
type="text" class="ElementStyle"/></TD><TD width="50%"
class="ElementStyle"><input value=""
name="XPostChange:/Quote/QuoteHeader/QuoteIssueDate"
type="hidden"/><input
onChange="javascript:setModified('XPostChange:/Quote/QuoteHeader/QuoteIss
ueDate');" size="15" value=""
name="XPostContent:/Quote/QuoteHeader/QuoteIssueDate" type="text"
class="ElementStyle"/></TD></TR><TR class="ListRowAlternate"><TD
class="StatusHeaderNew" width="50%" align="left">Quote Sent to Buyer:ABB-
1010</TD><TD class="StatusHeaderNew" width="50%" align="left">From
Seller:ABB-1010</TD></TR><TR><TD class="BodyText">ABB Cornerstone
Company</TD><TD class="BodyText">ABB Cornerstone
Company</TD></TR><TR><TD class="BodyText">45 Terrance Blvd.</TD><TD
class="BodyText">45 Terrance Blvd.</TD></TR><TR><TD
class="BodyText">7th Floor</TD><TD class="BodyText">7th
Floor</TD></TR><TR><TD class="BodyText">Auction Department</TD><TD

```

40

45

```


```

5

10

15

20

25

30

35

40

45



<OPTION VALUE=UnableToQuote > Unable To Quote  
 </SELECT></TD></TR><TR><TD class="StatusHeaderSmallNew"  
 valign="bottom" align="left"><span  
 class="RequiredField">\*</span>Item</TD><TD class="StatusHeaderSmallNew"  
 5 valign="bottom" align="left">Part#</TD><TD class="StatusHeaderSmallNew"  
 width="30%" valign="bottom" align="left">Description</TD><TD  
 class="StatusHeaderSmallNew" valign="bottom" align="left">Qty</TD><TD  
 class="StatusHeaderSmallNew" valign="bottom"  
 align="left">U/M</TD></TR><TR><TD class="BodyText" valign="top"  
 10 align="left"><input value=""  
 name="XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuoteItemDetail/LinItemNum/BuyerLinItemNum" type="hidden"/><input  
 onChange="javascript:setModified('XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuoteItemDetail/LinItemNum/BuyerLinItemNum');"  
 15 maxLength="11" size="5" value="1"  
 name="XPostContent:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuoteItemDetail/LinItemNum/BuyerLinItemNum" type="text"/></TD><TD class="BodyText"  
 valign="top" align="left">T-77-0-0-56</TD><TD class="BodyText" valign="top"  
 align="left" width="30%"/><TD class="BodyText" valign="top"  
 20 align="left">6000000.00</TD><TD class="BodyText" valign="top"  
 align="left">EA</TD></TR><TR class="ListRowAlternate"><TD  
 class="StatusHeaderSmallNew" colspan="5" align="left">To Be Completed by  
 Offeror:</TD></TR><TR class="ListRowAlternate"><TD colspan="5"><TABLE  
 border="0" cellpadding="0" cellspacing="1" width="100%"><TR><TD  
 25 class="StatusHeaderSmallNew" valign="bottom" align="left"><span  
 class="RequiredField">\*</span>Unit Price:</TD><TD  
 class="StatusHeaderSmallNew" valign="bottom" align="left"><span  
 class="RequiredField">\*</span>U/M:</TD><TD class="StatusHeaderSmallNew"  
 valign="bottom" align="left"><span class="RequiredField">\*</span>Total  
 30 Price:</TD></TR><TR><TD valign="top" align="left"><input value=""  
 name="XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingDetail/PricingDetail/ListOfPrice/Price[1]/UnitPrice/UnitPriceValue"  
 type="hidden"/><input  
 onChange="javascript:setModified('XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingDetail/PricingDetail/ListOfPrice/Price[1]/UnitPrice/UnitPriceValue');"  
 35 name="XPostContent:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingDetail/PricingDetail/ListOfPrice/Price[1]/UnitPrice/UnitPriceValue" type="text"  
 class="ElementStyle"/></TD><TD valign="top" align="left"><input value=""  
 name="XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingDetail/PricingDetail/ListOfPrice/Price[1]/UnitPrice/UnitOfMeasurement/UOMCoded"  
 type="hidden"/><SELECT  
 onChange="javascript:setModified('XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingDetail/PricingDetail/ListOfPrice/Price[1]/UnitPrice/UnitOfMeasurement/UOMCoded');"  
 40 height="10"  
 name="XPostContent:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingDetail/PricingDetail/ListOfPrice/Price[1]/UnitPrice/UnitOfMeasurement/UOMCoded"  
 "><OPTION VALUE= SELECTED >  
 <OPTION VALUE=FC > 1000 Cubic Feet

<OPTION VALUE=MQ > 1000 Meters  
 <OPTION VALUE=KS > 1000 Pounds Per Square Inch  
 <OPTION VALUE=BP > 100 Board Feet  
 <OPTION VALUE=KK > 100 Kilograms  
 5 <OPTION VALUE=YL > 100 Lineal Yards

\*\*\*

10 <OPTION VALUE=B22 > Kiloampere  
 <OPTION VALUE=TAH > Kiloampere Hour Or Thousand Ampere Hour  
 <OPTION VALUE=B24 > Kiloampere Per Metre  
 <OPTION VALUE=B23 > Kiloampere Per Square Metre  
 <OPTION VALUE=KBA > Kilobar  
 <OPTION VALUE=2Q > Kilobecquerel  
 15 <OPTION VALUE=B25 > Kilobecquerel Per Kilogram  
 <OPTION VALUE=2P > Kilobyte  
 <OPTION VALUE=KB > Kilocharacter  
 <OPTION VALUE=B26 > Kilocoulomb  
 <OPTION VALUE=B27 > Kilocoulomb Per Cubic Metre  
 20 <OPTION VALUE=B28 > Kilocoulomb Per Square Metre  
 <OPTION VALUE=2R > Kilocurie  
 <OPTION VALUE=B29 > Kiloelectronvolt  
 <OPTION VALUE=78 > Kilogauss  
 <OPTION VALUE=KGM > Kilogram

\*\*\*

<OPTION VALUE=WG > Wine Gallon  
 <OPTION VALUE=WM > Working Month  
 30 <OPTION VALUE=WR > Wrap  
 <OPTION VALUE=YRD > Yard  
 <OPTION VALUE=ANN > Years  
 </SELECT></TD><TD valign="top" align="left"><input value=""  
 name="XPostChange:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingD  
 35 etail/PricingDetail/TotalValue/MonetaryValue/MonetaryAmount"  
 type="hidden"/><input  
 onChange="javascript:setModified('XPostChange:/Quote/ListOfQuoteDetails/Quo  
 teDetails[1]/QuotePricingDetail/PricingDetail/TotalValue/MonetaryValue/Monetar  
 yAmount');" maxLength="28" size="15" value=""  
 40 name="XPostContent:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotePricingD  
 etail/PricingDetail/TotalValue/MonetaryValue/MonetaryAmount" type="text"  
 class="ElementStyle"/></TD></TR></TABLE></TD></TR><TR  
 class="ListRowAlternate"><TD class="StatusHeaderSmallNew"  
 align="left">Attachment:</TD><TD colspan="4" class="ListRowAlternateText"  
 45 align="right"><input value=""  
 name="XPostDefaultValue:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuotelItem  
 ListOfAttachment/ListOfAttachment/Attachment[1]/AttachmentLocation"  
 type="hidden"/><input value=""  
 name="XPostAddInto:/Quote/ListOfQuoteDetails/QuoteDetails[1]"

```

type="hidden"/><input
onClick="javascript:doXPostProcessWithDefault('CRSaveAction',
'XPostAddInto:', '/Quote/ListOfQuoteDetails/QuoteDetails[1]',
'QuoteItemListOfAttachment',
5 'XPostDefaultValue:/Quote/ListOfQuoteDetails/QuoteDetails[1]/QuoteItemListOfA
ttachment/ListOfAttachment/Attachment[1]/AttachmentLocation',
'urn:attachment1_1'); return false;" class="ElementStyle1"
name="AddAttachment" value="Add Attachment"
type="submit"/></TD></TR><TR class="ListRowAlternate"><TD height="8"
10 colspan="5"></TD></TR><TR bgcolor="#000000"><TD height="1"
colspan="5"></TD></TR></TABLE><BR xmlns:cbl="urn:x-
commerceone:document:com:commerceone:XCBL30:XCBL30.sox$1.0"
xmlns:xm="http://www.commerceone.com/xcbllmailbox"
20 xmlns:ext="http://www.commerceone.com/xmlpres"/><TABLE xmlns:cbl="urn:x-
commerceone:document:com:commerceone:XCBL30:XCBL30.sox$1.0"
xmlns:xm="http://www.commerceone.com/xcbllmailbox"
xmlns:ext="http://www.commerceone.com/xmlpres" width="100%" border="0"
cellspacing="0" cellpadding="0"><TR class="ListHeader"><TD
25 class="StatusHeaderNew1" align="left">Quote Summary:</TD></TR></TABLE>
</Frame>

</td>
</tr>
30 </table>

<!-- END MAIN document area -->

</FORM>
35 <!-- END FORM -->

<!-- END MAIN content cell -->

40 </TD></TR></TBODY>
</TABLE>
</BODY>
</HTML>
<!-- ----- end Mailbox20.jsp ----- -->
45

```

[0056] Figure 13 is a sample user interface screen. This screen corresponds to the HTML code above. Alternative embodiments may use a display language other than

HTML or may use smart terminals or thin clients instead of being specifically adapted to browsers. In figure 13, a number of user controls are provided. A send control 1301 allows a user to send a completed document. Preferably, the document is validated prior to the send control being functional. A save as draft control 1302 allows the user to save the document without sending it or validating it. A validate control 1303 allows the user to invoke one or more levels of validation. Data entry validation may be supported for editable fields without invoking this validate control. For instance, numeric data may be required in the unit price field 1328 and the total price field 1331, even without invoking the validate control 1303. Controls for adding and removing line items 1304 are appropriate to a quote, request for quote or similar electronic commerce document. These controls may either invoke a procedure running within a browser, such as a JavaScript procedure, or may cause a client to contact a server for an updated screen. An add attachment control 1305 allows the user to add an attachment, which then is listed under attachments 1332. In this example, the fields of a quote can be grouped. The status of this example is draft 1311. A reply to designee is identified 1312. The document type is a quote 1313. The quote header 1314 in this example includes a sequence number, which may be automatically generated by either the client or server, and a date field which may be automatically populated or manually entered. The issuer and recipient data 1315 is generated by a transformation of a request for quotation. The quote detail 1321 includes several fields of various types. A status field populated from a pull down menu as provided 1322. An item number 1323 is given a starting value, which may be editable. Additional fields 1324-1327 copy into the quotation information from the request for quotation. The user completes the unit price 1328. A unit of measure is selected from a pull down menu 1329. This field could be given a starting value to match the request for quotation. A total price field 1331 can be completed by the user or automatically calculated as an extension of the quantity and unit price. A quote summary field 1333 can be generated automatically or filled in by a user. Not visible in the sample screen, but included in the HTML code above are path specifications corresponding to fields of data that appear on the screen. Path specifications may be provided for all fields or only for editable fields. Also not visible are update status fields. The HTML code above provides a hidden update status field for at least the editable fields. This optional update status field can be used by either the client or the

server to process and update more efficiently. The client can use an update status field to limit the information communicated to the server, in some embodiments. The server can use the update status field to avoid changing values already resident in memory, in some embodiments.

5 [0057] Figure 14 is a sequence diagram for communications among components such as those depicted in the alternative embodiments of figures 6, 7 and 11. The browser 1401 is the component with which the user interacts. It may be a web browser, a thin client, a smart terminal or other equivalent hardware and software interface. The servlet 1402 runs on the server. The rule selector 1403 is a component  
10 for choosing the appropriate rules, forms etc. The transformation component applies a declarative transformation to a self-describing, structure document. For instance, an XSL style sheet can be applied to an XML document to generate a new XML document. Or, a JSP page or XSLT style sheet can be applied to an XML document to generate an HTML page for display. A variety of declarative transformations can  
15 be used. A database 1405 manages persistent storage. An XML engine 1406 manipulates an XML document in memory.

[0058] The sequence in figure 14 begins at the browser 1401, with a user viewing a request for quotation document 1411. The user requests generation of a reply document 1412. The servlet initiates mapping the document to a set of potential  
20 replies 1413. The servlet may receive from the browser a starting document type or it may receive a document ID for a self-describing, structured document and learn the document type from accessing the document. From a starting document type, the rule selector is invoked 1414 to determine the available reply types. The transformation engine generates an interface screen 1415, which the servlet returns to the browser  
25 1416. The user selects a reply type, for instance a quote 1417. The servlet transforms the starting document, in this example a request for quotation, into a draft resulting document 1418, in this case a quote. In no particular order, a style sheet is retrieved 1419 and applied to the quote to generate a user interface screen (implied), the quote is persisted by the database 1421 and, in some embodiments, the quote is loaded into  
30 an internal data structure for ease of manipulation 1421. This data structure may be a DOM tree. Once data for the user interface has been generated, it can be transmitted by the servlet to the browser 1422. The user updates fields and sends all or part of the data received from the servlet back 1423. The servlet may either have the draft quote

in memory, it may retrieve the draft quote from the database 1424 and create a data structure in memory 1425 to be updated 1426, or it may create a quote from the data received back from the browser. In the embodiment illustrated here, the draft quote is retrieved from a database and updated 1426 with the fields that have been flagged as modified by the user. The updated data structure in memory is used to generate a self-describing, structured document 1427, which is persisted to the database 1428. Once updating has been accomplished, the update is communicated to the browser 1429. Not shown in this sequence are the validation steps previously discussed.

[0059] Figure 15 depicts an action command framework. In this framework, the servlet translates actions into one or more commands. This facilitates modification of process flow and reuse of sub process steps. A number of software components 1501-1507 are used to implement this framework. The sequence diagram depicts initialization of the object factory 1511. A request container is created 1512. The requested action from a user is created using the initialized object factory and request container 1513. The object factory acts to create the requested action 1514, consistent with appropriate selection of parameters, forms, etc. The servlet directs execution of the action 1515. A set of commands are created to carry out the action 1516. The object factory creates the appropriate commands 1517. The commands are executed 1518, and an appropriate JSP page accessed 1519.

[0060] Figure 16 depicts interaction among the servlet, the XPost interpreter and the XML engine when receiving a request from the client. A set of software components 1601-1616 are provided. Prior to the sequence illustrated in this figure, the servlet 1601 receives data including sets of update operators, path specifications and update values. In one embodiment, the servlet receives this data from a client browser via an HTTP post command in an HTML format. The servlet passes the data 1611 to an iterator 1602. The iterator builds a dictionary, table or other data structure in a form that the interpreter expects 1612. The servlet 1601 invokes 1613 interpreter 1603 to process the data structure. The interpreter exchanges messages 1614, 1615 with access control to determine the security status of the requestor. Assuming that the security check passes, the interpreter instantiates 1616 the resources needed to process the data. Fields received from the user are unmasked 1617 from a user-friendly format to a canonical format used for internal processing. This un-masking is repeated as often as required 1618. With data in a canonical format, the XML engine

1605 is invoked 1619 to update a document consistent with the data structure. The updated document is returned 1620, 1621. Some of the components involved in the sequence are depicted in the block diagram of figure 17. Updated HTML 1701 is posted via HTTP to the Web server 1702. The XPost interpreter 1703 acts upon  
 5 XPost commands which may be stored in a hash table, dictionary or other data structure 1704. The XPost interpreter 1703 controls updating of an XML instance 1712 in a database 1711, which manages persistent storage.

[0061] Figure 18 is a simplified flow chart for a method of updating a self-describing, structured document. The character string 1801 is received. The

10 character string may be encoded according to any of a variety of standards, such as UTF-8. The character string includes an update operator or other indication of the desired operation, a path specification identifying a node or location within a self-describing, structured document, at which the update operator is to be applied, and one or more update values. One update operator may apply to a plurality of path  
 15 specifications and update values. Alternatively, the update operator may be received as part of the URL that invokes processing or it may be implied by the manner in which the update component is invoked. Update operators may include deleting an element, adding an element before an identified node as a sibling of the node, adding an element after the identified node is a sibling of the node, adding the element as a  
 20 child of the identified node, or a combination of delete and add to accomplish a move or a replace. Other update operators can be supported. The path specification may be compliant with XPath or a subset of XPath. It is preferred to fully specify the path in absolute, instead of relative terms. Reference to identifying a node in the self-describing, structured document is not meant to imply that the document needs to be  
 25 manipulated using the document object model or any other tree structured tool. The document can readily be manipulated with SAX-compliant tools or other event-driven tools. The character string is parsed 1802. A self-describing, structured document is accessed 1803. The document has a document type, which may be received or may be obtained from the document itself. The document may be accessed from disk 1811  
 30 or may already be resident in memory when the character string is received. XML documents are among those that can be considered self-describing, structured documents. A schema corresponding to the document type of the document is accessed 1804. The schema may be accessed from a database 1812, disk or memory.

The schema is used for validating 1804 application of the update operator and update values at the node identified in the past specification. The schema may be SOX-compliant, XML schema-compliant, or RELAX-compliant. Other schemas may apply to XML or non-XML documents. Following successful validation, the

5 document is updated 1805 with update values at the node specified in the past specification. In some embodiments, the character string will include a document ID or a binary document ID will be sent in addition to a character string. In other implementations, the document ID will be implied by prior state information, in which case the document may already be in memory. All or part of the document

10 may be accessed via a data-object-model data structure in memory. Any of the embodiments may be used with the past specification that is compliant with any version of an XPath standard. Similarly, any of the embodiments may be used with a schema that is compliant with any version of a Sox standard. In some embodiments, an element set will be constructed, such as a set of fields for a line item in a purchase

15 order. A set of elements associated with a single location or node may be assigned a shorthand or alias and accessed or specified using the shorthand or alias. When an alias is used for a set of elements, more than one update value may be associated with a single pass specification. In addition to validation against the schema, this process may further involve validation against a business processing rule set. The business

20 processing rules may be selected according to the document type, trading partner, or other criteria. The business processing rules may be implemented by a set of Schematron declarations or may be implemented in procedural logic.

**[0062]** Figure 19 illustrates a protocol for updating a self-describing, structured document corresponding to one of a plurality of document schemas. The entities

25 represented in this diagram are a requestor 1901, a respondent 1902 and a source of declarative transformations 1903. The protocol begins with receiving of request 1911 identifying a starting document and specifying a document type to be generated from the starting document. The identification of the starting document and the specification of the document type may be received in two separate messages, so that

30 the starting document is known from prior state information when the type of document is specified. The respondent, which may be a servlet running on a server, accesses one or more declarative transformations 1912, 1913 corresponding to the starting document and the specified document type. A rule selector may determine



which declarative transformations are accessed. The system applies the declarative transformations to the starting document 1914, producing a plurality of path specifications for fields corresponding to the specified document type, starting values based on the starting document for at least some of the fields in some applications, fields to be completed by the user. The application of declarative transformations may produce a draft resulting document, a form, or both. In one embodiment, the system sends a form responsive to the request 1915 to the user, receives an updated form 1916, validates the update information 1917 and generates a final resulting document 1919. Validating the update information may involve one or more types of validation, such as validation against the schema and validation against business processing rules.

**[0063]** Figure 20 is a flowchart overview of validation. A document 2001 is received 2002. Optionally, a set of declarative rules 2011 for validation is accessed 2003. The declarative rules may include schema level validation based on a Sox schema and business logic level validation using Schematron. If validation fails on the schema level, field validation errors may be displayed without proceeding to Schematron validation. Validation errors may be displayed adjacent to the node at which the error was detected. If validation fails on the Schematron level, error messages may be displayed at the top of the user interface or wherever appropriate, using the XSLT document () function. The default.xsl sample code set forth above includes logic for processing both Schematron and field validation errors.

**[0064]** One or more errors in the primary document are detected 2004. A secondary self-describing, structured document 2012 including the detected errors is generated. For the detected errors, an error identifier and a path specification identifying a node within the primary document corresponding to one or more of the detected errors are generated. The primary 2001 and secondary 2012 documents are merged 2005 for display 2006. In one embodiment, the secondary document is an XML document. An enhanced version of the XSLT document () function is used to allow retrieval of the secondary document containing error text. The XMLPres component, which implements the XSLT functionality, is an enhanced version of Apache Foundation's Xalan. This component takes into account various concepts such as name spaces in Sox documents and polymorphism. Extension of the

document () function enables the loading of an external document to be specified by the invoker.

**[0065]** Figure 21 is a flowchart overview of a method of searching a plurality of self-describing structure documents, the documents including document fields. A

5 server 2101 provides a user Interface 2102. The user interface includes, in one or more screens, the document type selection filter and one or more document fields selection filters. These document field selection filters are context sensitive to the selected type of document, if a document type is selected. One or more value specification fields are supplied, to be filled in for the field selection filters. These

10 value specification fields are context sensitive, so that field format errors, such as text characters in a numeric field, can be detected directly at the user interface, without return to the server. In addition, has non displaying fields, one or more path specifications corresponding to the document fields and the value specification fields are provided. The path specifications identifying nodes to be tested against the

15 completed value specifications for a search. The server next receives data specifying a search 2104. This data typically comes from a user responding to the user interface. It includes the selected document type, completed value specifications and path specifications for the selected fields. Alternatively, aliases may be given for selected fields, which are expanded into path specifications by reference to a data structure

20 2103. A search is conducted through a subset of self-describing, structured documents 2105, based on the completed value specifications and the corresponding path specifications. The subset includes documents of the selected document type. From the perspective of the user, the user interface displays a document type selection filter, one or more document field selection filters, which are context sensitive to a

25 selected document type and one or more value specification fields, which are context sensitive to the document fields. Not displayed to the user, except perhaps a debugging mode, are path specifications corresponding to the document fields and the value specification fields. The path specifications identify nodes to be tested against completed value specifications when a search is conducted. The user selects a

30 document type and one or more document fields appropriate to the document type. Value specifications are completed by the user. The user's selections are transmitted to a server which may then conduct a search. Recast as a device, the computer user interface includes a document type selection filter, one or more document field

selection filters coupled by context to a selected document type, one or more value specification fields, coupled by context to selected document fields, and, has non displaying fields, one or more path specifications corresponding to the document fields and to the value specification fields.

- 5 [0066] A context-based search facility is user configurable. Database configuration interfaces may be used for specifying those fields to be indexed. The user interface can be configured to provide user-friendly names to be displayed corresponding to XPath or field element name representations. Configuring the user interface permits fields that have different names in different document types to begin  
10 a common, user-friendly name at the user interface. Figure 22 identifies components of context-based searching. Upon receiving a new document 2211, the generic document handling service 2212 will scan its configurations 2201 to determine if 2213 the document has been configured for content-based searching. If the document type is configured to be searched, the system identifies fields to be indexed. In one  
15 embodiment, it uses the XPath specification of the incoming document fields to locate the indexable fields. After indexing, a process is invoked to persist the data 2221. Persistent indexed data is stored 2222. Further processing proceeds. Database configuration is supported by various tables.

- [0067] Figure 23 depicts a document type list used for database search  
20 configuration. A list, table or other data structure may be used to capture all document types recognized by the generic document handling system, or at least by the indexing system. The type ID column 2301 contains an integer value for an automatically generated field ID. This unique field ID is used as a shorthand for the document type. The type column 2302 is the character string identifying the  
25 document type. For instance, "request for quotation" is a document type. The name space column 2303 identifies the name space to which this document type belongs. The name space may be a uniform resource name that fully qualifies a location at which a schema for the document can be found. Column 2304 is an extra column reserved for future use. The application-created column 2305 is an integer used to  
30 flag fields in which data is entered manually. This field should be left null if data is entered manually.

[0068] Figure 24 depicts a field definition table that is populated with path specifications of elements that will be searched. The field definition column 2401 is

an automatically generated integer value, that is used as a shorthand for the field. The MPID column 2402 is a unique identifier for a particular trading partner for whom the past specification that follows is valid. The type ID column 2403 is an integer value mapped to the document's type ID 2301. Many fields can be mapped to a single document type. The name field 2404 is a path specification. In some embodiments, it is an XPath for an XML document.

**[0069]** XPath provides a unique way to identify a field inside a XML document, but it may not be meaningful to an end user. As illustrated in Figure 25, an alias to the XPath may be used as an alternative to embedding the full XPath string with each field used in the interface. The alias may be displayed in the user interface as a field name. The generic document handling system will associate the alias with the full XPath string when user submits a request. The segment 2501 is an example of XPath and alias, segment 2502 is another example of multiple occurrence XPath and alias, segment 2503 is another example of different name space XPath and alias.

**[0070]** Figure 26 is a table of searchable data extracted for indexing. The searchable field column 2601 identifies the field in which the data occurs. The document sequence field 2602 corresponds to the document in which the field was found. The field definition column 2603 corresponds to the field definition IDs in figure 24. The value column 2604 contains a searchable value. This value may be a single token or it may be the full contents of the field. A single token may, in some embodiments, include embedded blanks or other non-text characters.

**[0071]** While the preceding examples are cast in primarily in terms of a method, devices and systems employing this method are easily understood. An electronically readable medium, such as a DVD, CD, memory module, magnetic disk or magnetic tape containing a program capable of practicing aspects the claimed method is one such device. A computer system comprising one or more servers and/or workstations having memory loaded with a program practicing the claimed method is another such device.

**[0072]** While the present invention is disclosed by reference to the preferred embodiments and examples detailed above, it is understood that these examples are intended in an illustrative rather than in a limiting sense. It is contemplated that modifications and combinations will readily occur to those skilled in the art, which

CMRC 1009-1

modifications and combinations will be within the spirit of the invention and the scope of the following claims.

[0073] We claim as follows:

1009631-124804